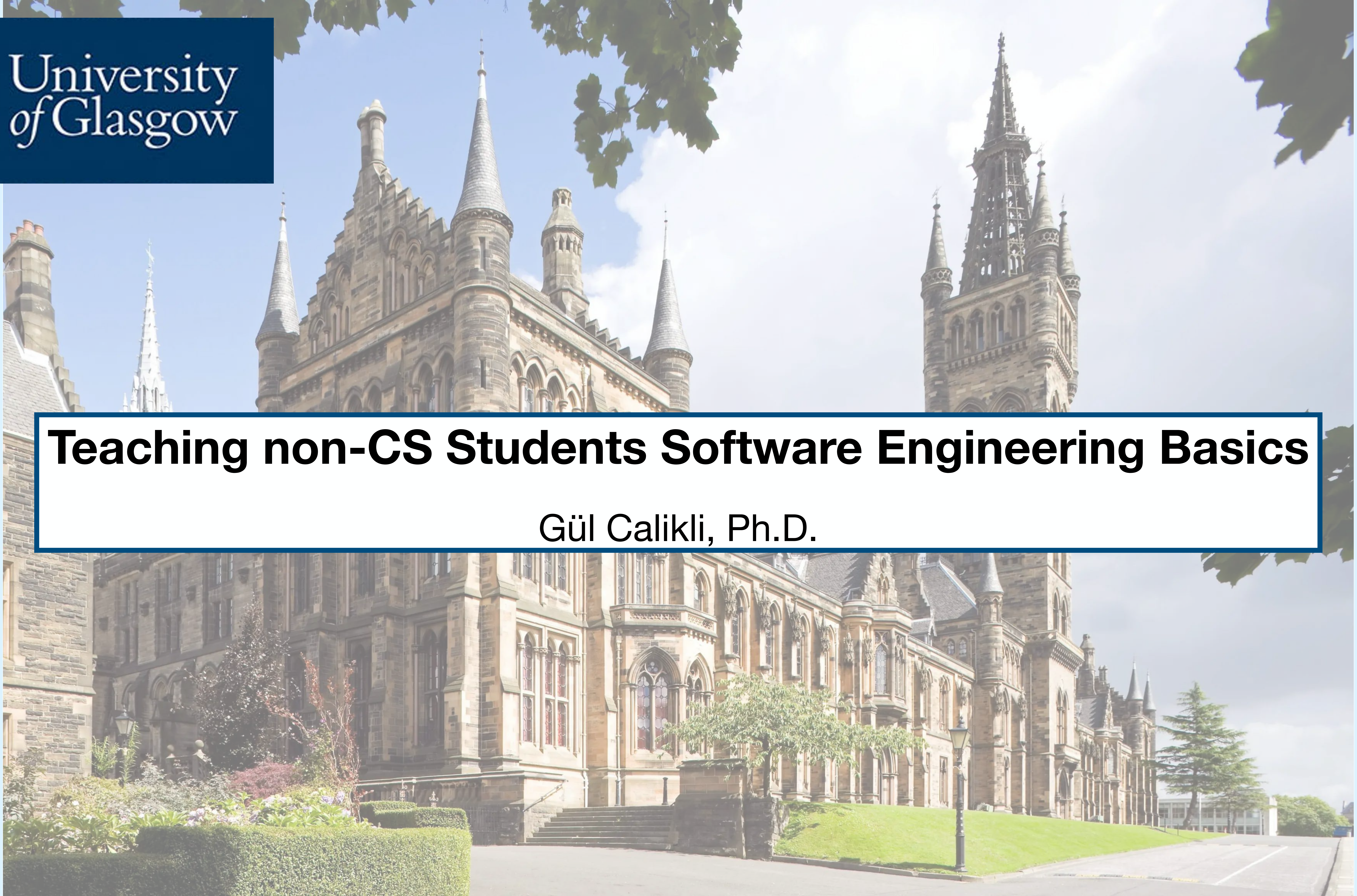


University of Glasgow

Teaching non-CS Students Software Engineering Basics

Gül Calikli, Ph.D.

Software Engineering IT “SEIT” Course

- A course offered by Information Technology (IT) Postgraduate Program;
- Students taking the course come from disciplines **other than** Computer Science and Engineering;
- Aims to teach basic concepts of software engineering;
- ~310 students enrolled (for the academic year 2022/2023).
- Course runs throughout two semesters spread throughout the entire academic year.

SEIT Course Intended Learning Outcomes (ILOs)

- ☐ Carry out a requirements analysis and write requirements specification;
- ☐ Create UML diagrams which model aspects of the domain and the software solution;
- ☐ Apply design principles and patterns while designing and implementing simple software systems;
- ☐ Carry out software testing;
- ☐ Apply project management techniques.

SEIT Course Intended Learning Outcomes (ILOs)

☐ Carry out a
specificat

☐ Create U
and the s

includes

Coupling types:

- content coupling
- control coupling
- stamp coupling
- data coupling
- ...

Cohesion types:

- Coincidental/utility cohesion (usually bad)
- Sequential/procedural/temporal cohesion
- Functional cohesion (best)
- ...

☐ Apply design principles and patterns while designing and implementing simple software systems;

☐ Carry out software testing;

☐ Apply project management techniques.

Refactoring Code (Low Coupling & High Cohesion)

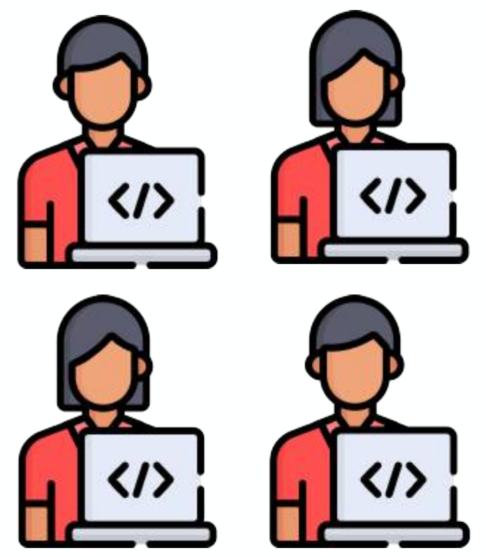
How did we teach refactoring code for low coupling and high cohesion?

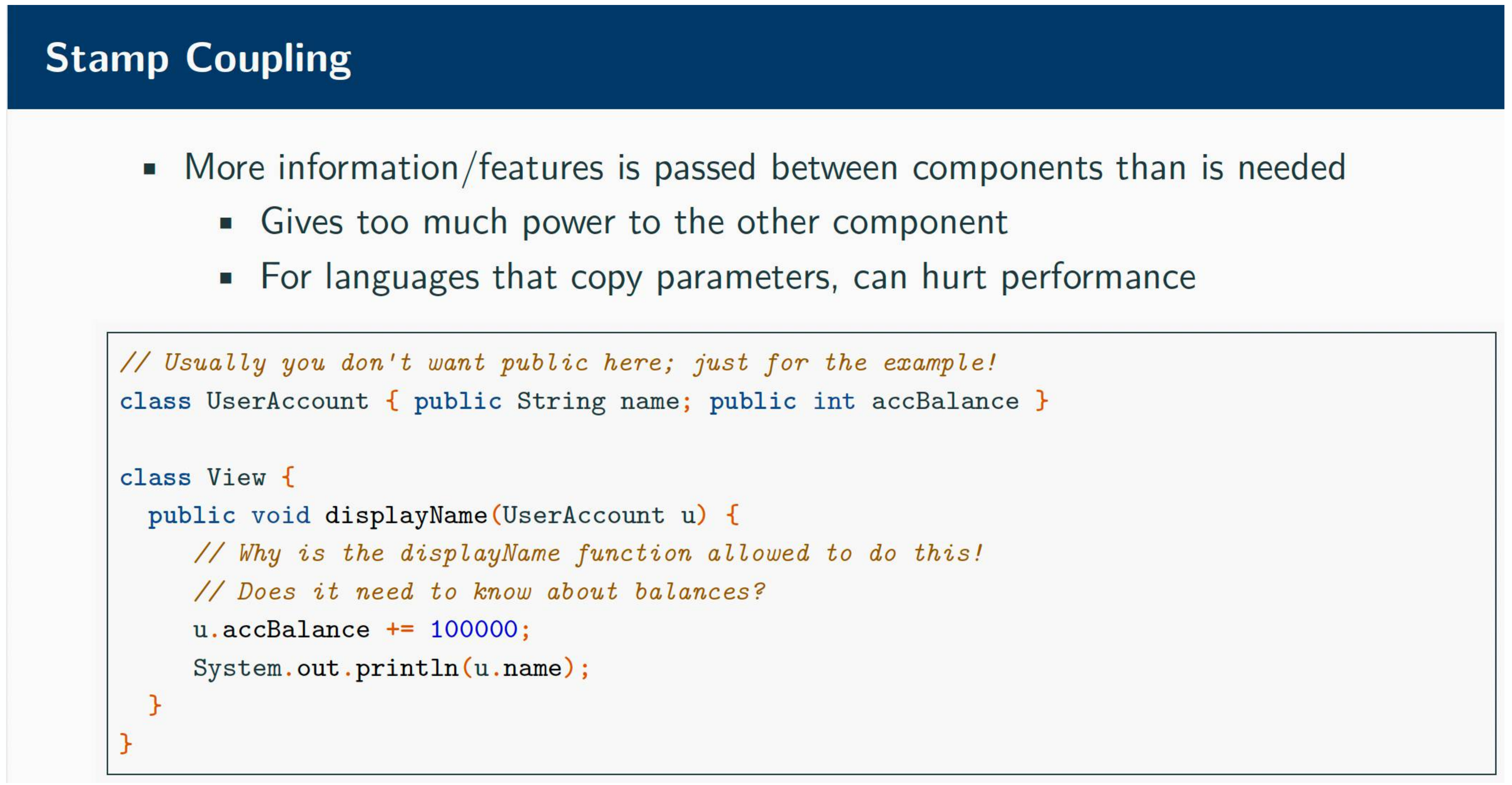
Refactoring Code (Low Coupling & High Cohesion)

Fall 2021 - Spring 2022
......
2021
2022
2023
2024

Online Lectures

Coding Labs



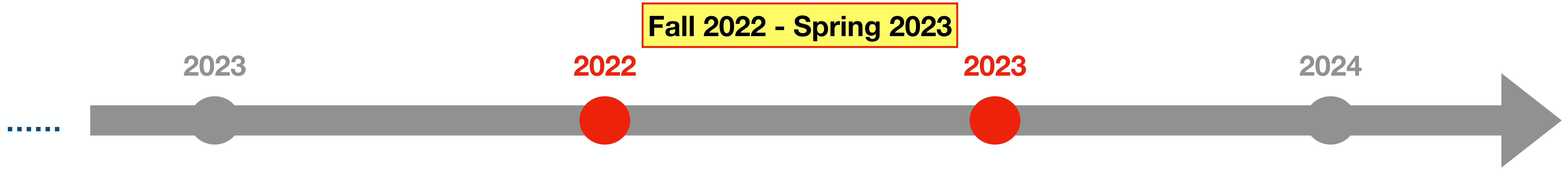


Stamp Coupling

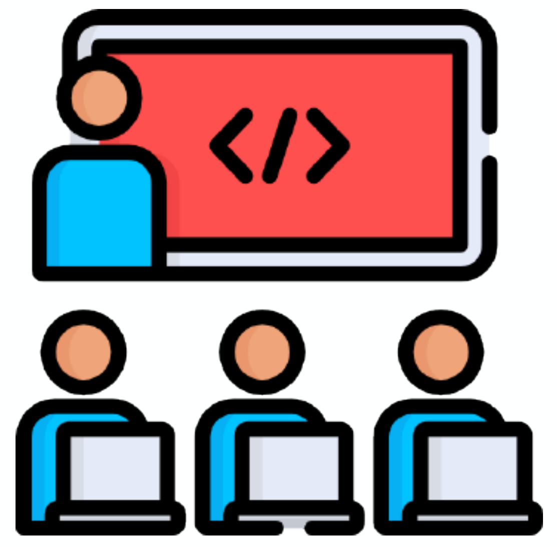
- More information/features is passed between components than is needed
  - Gives too much power to the other component
  - For languages that copy parameters, can hurt performance

```java
// Usually you don't want public here; just for the example!
class UserAccount { public String name; public int accBalance }

class View {
  public void displayName(UserAccount u) {
    // Why is the displayName function allowed to do this!
    // Does it need to know about balances?
    u.accBalance += 100000;
    System.out.println(u.name);
  }
}
```

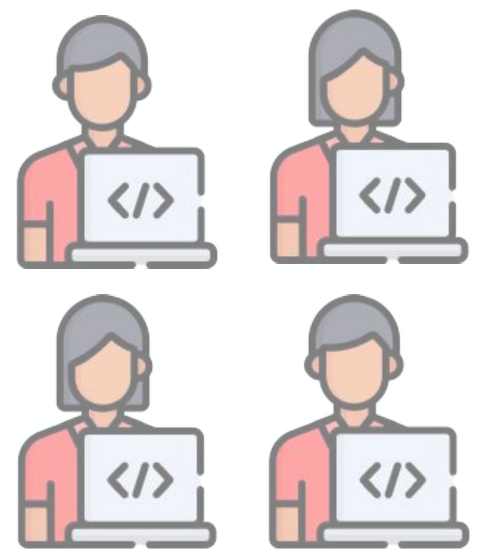
Refactoring Code (Low Coupling & High Cohesion)



Live Refactoring Session



Coding Labs



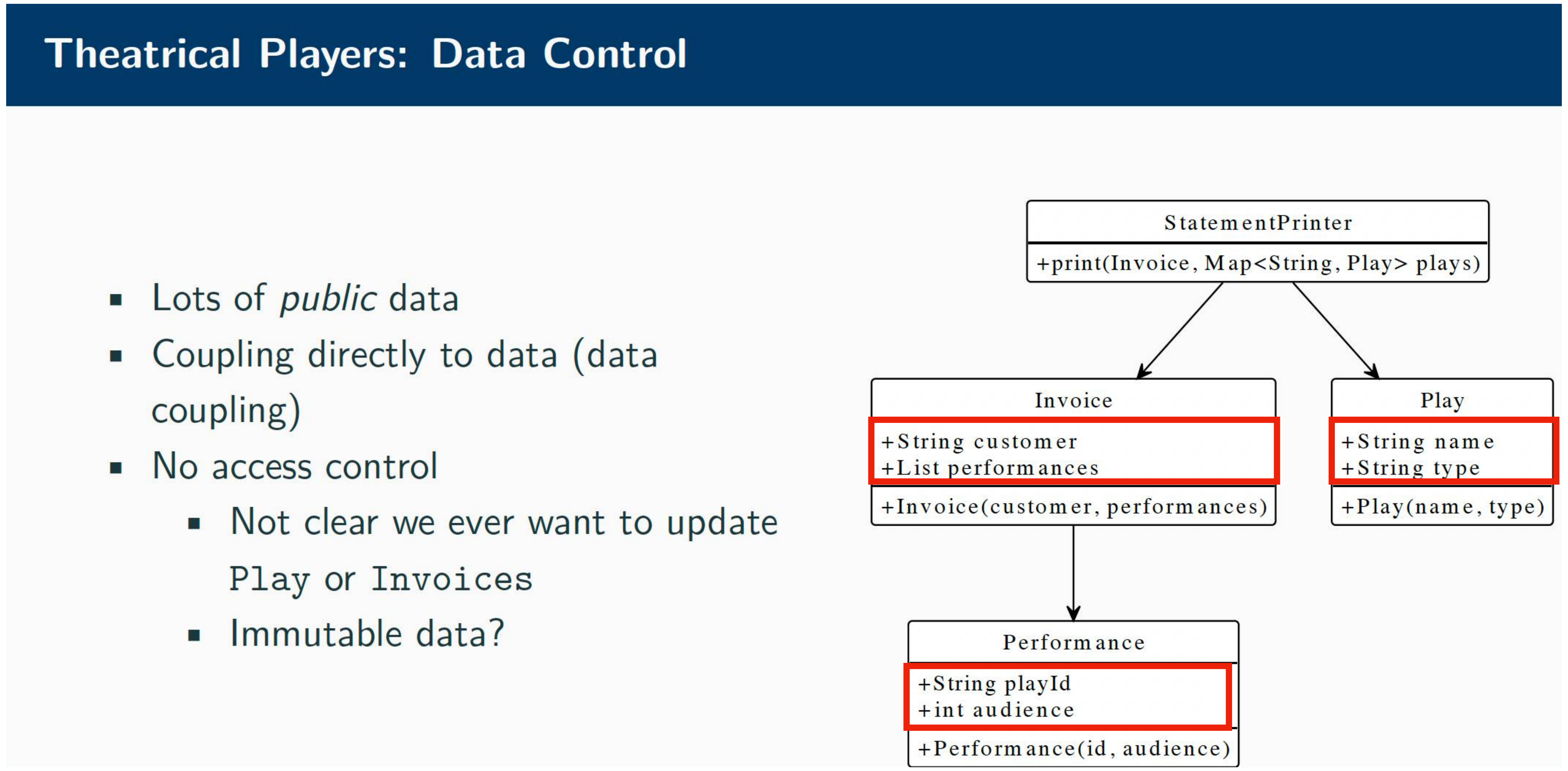
Live Refactoring Session

The teachers improves some code and students follow:

- An example based on “Theatrical Players Refactoring Kata”
  - Code to generate invoices for companies attending plays
  - Code Katas are exercises to improve SE skills and learn new techniques
  - Available online: https://github.com/ythirion/Theatrical-Players-Refactoring-Kata

Live Refactoring: Example Improvements

Identify and remove **content coupling**



Theatrical Players: Data Control

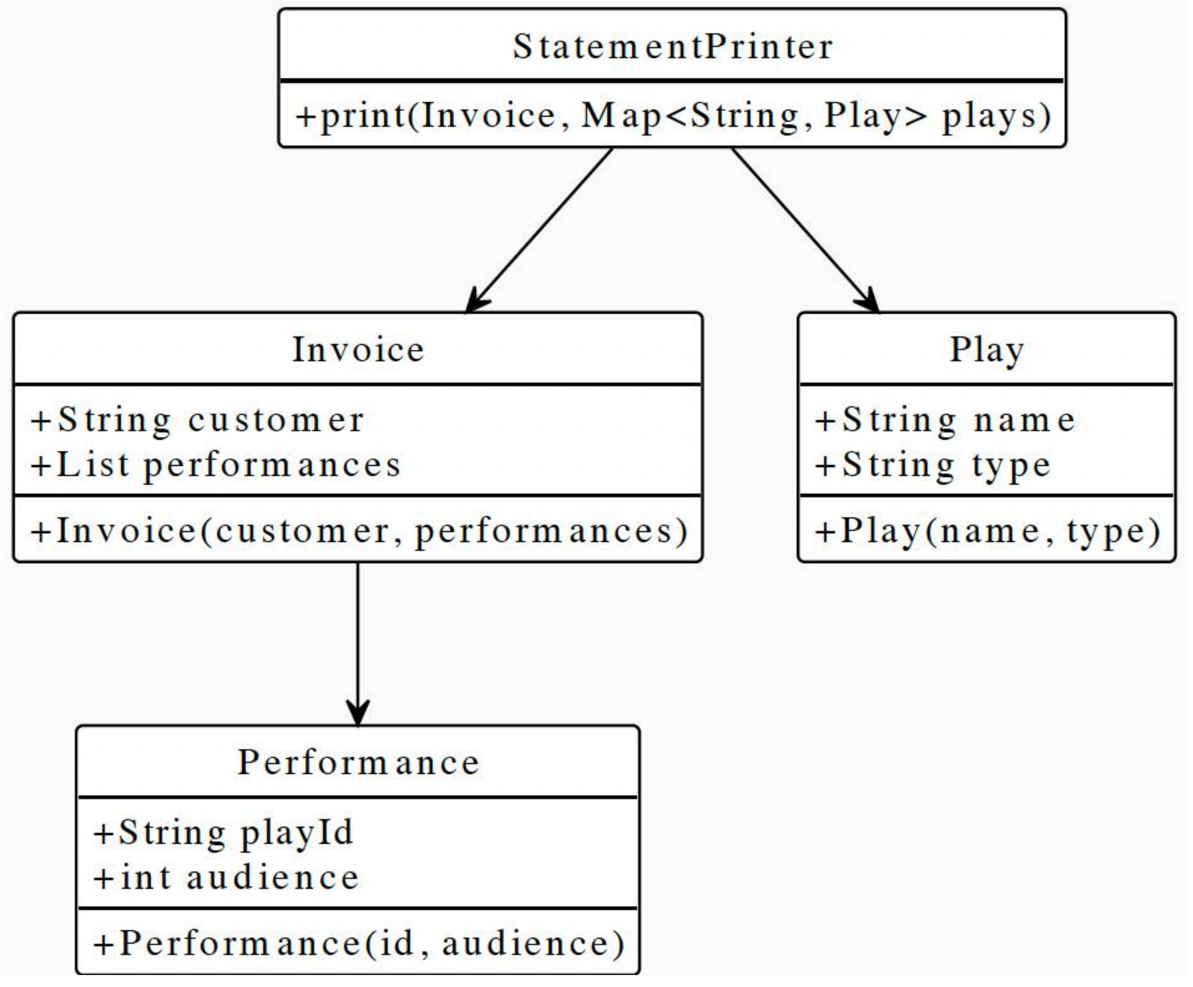
- Lots of *public* data
- Coupling directly to data (data coupling)
- No access control
  - Not clear we ever want to update `Play` or `Invoices`
  - Immutable data?

Live Refactoring: Example Improvements

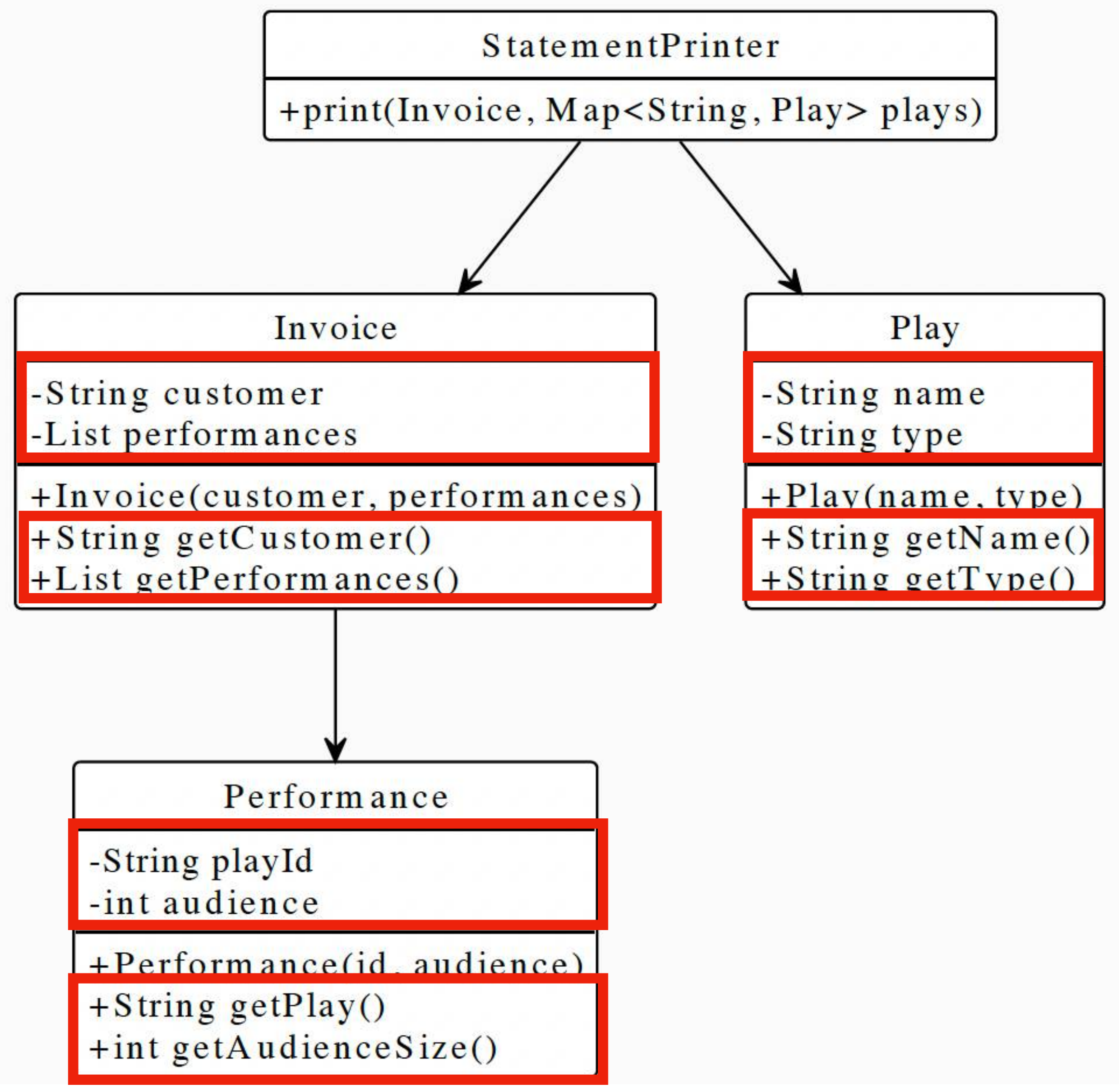
Identify and remove **content coupling**

Theatrical Players: Data Control

- Lots of *public* data
- Coupling directly to data (data coupling)
- No access control
  - Not clear we ever want to update `Play` or `Invoices`
  - Immutable data?



Theatrical Players: Data Control via Getters

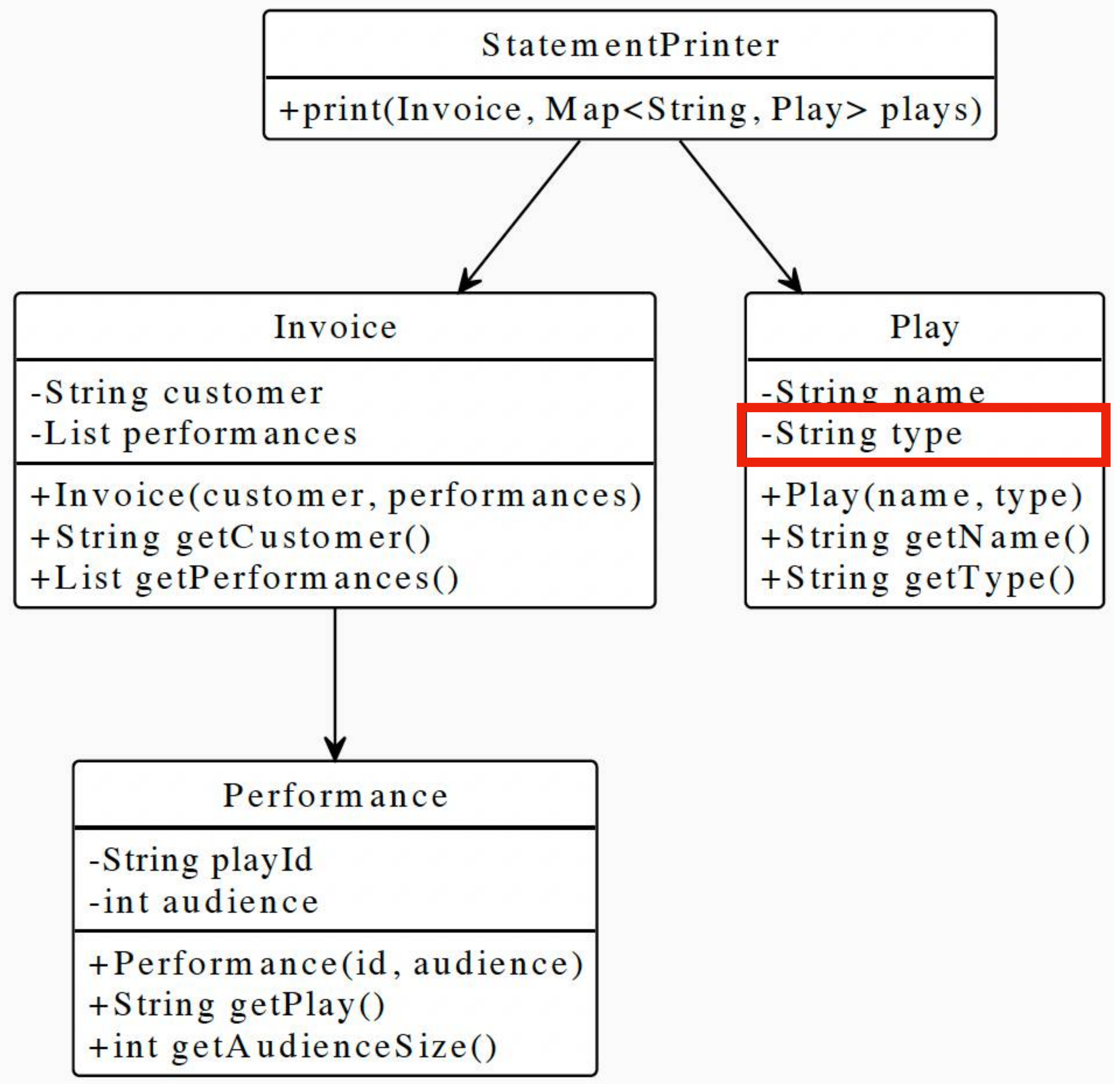


Live Refactoring: Example Improvements

Identify and remove **control coupling**

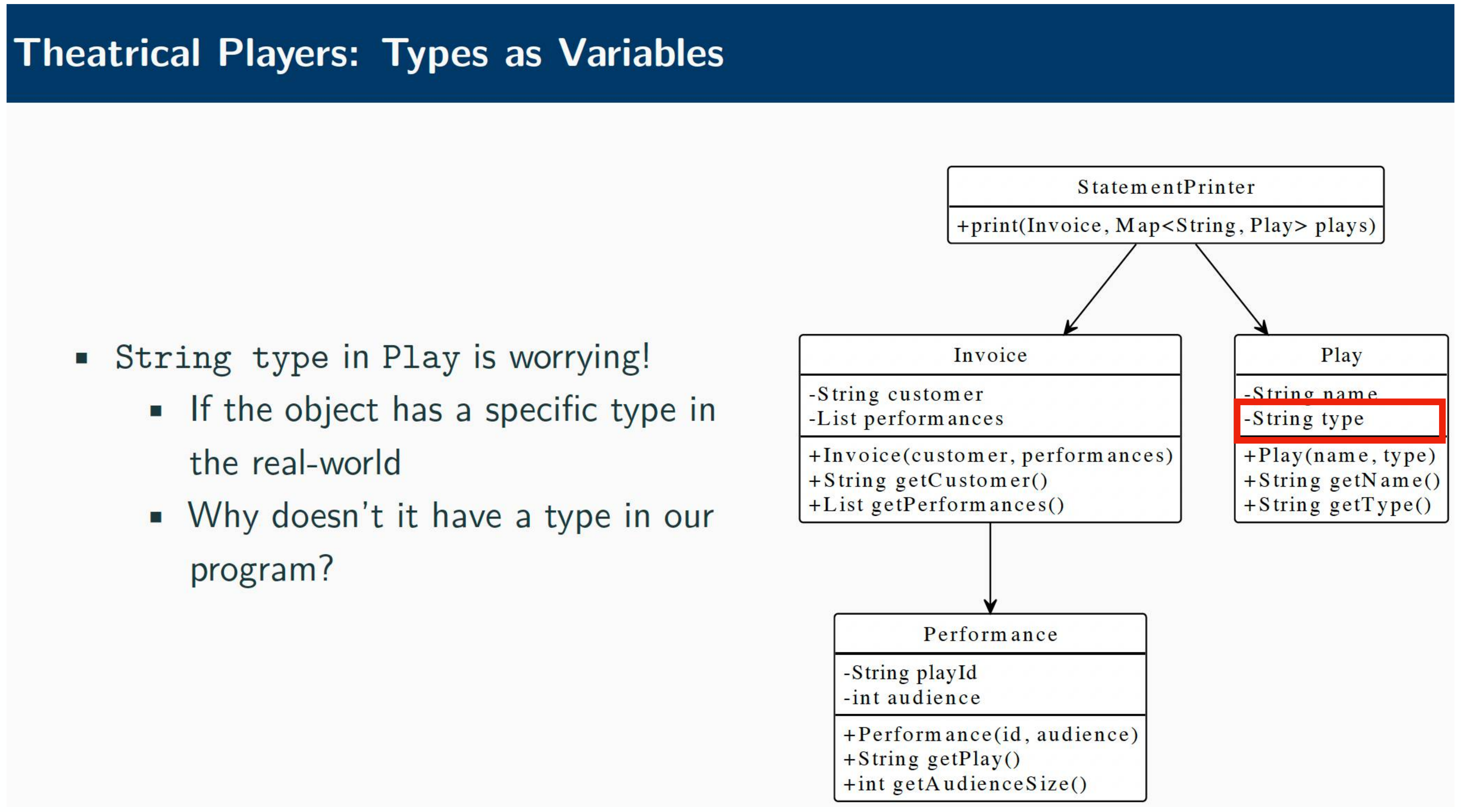
Theatrical Players: Types as Variables

- String type in Play is worrying!
  - If the object has a specific type in the real-world
  - Why doesn't it have a type in our program?



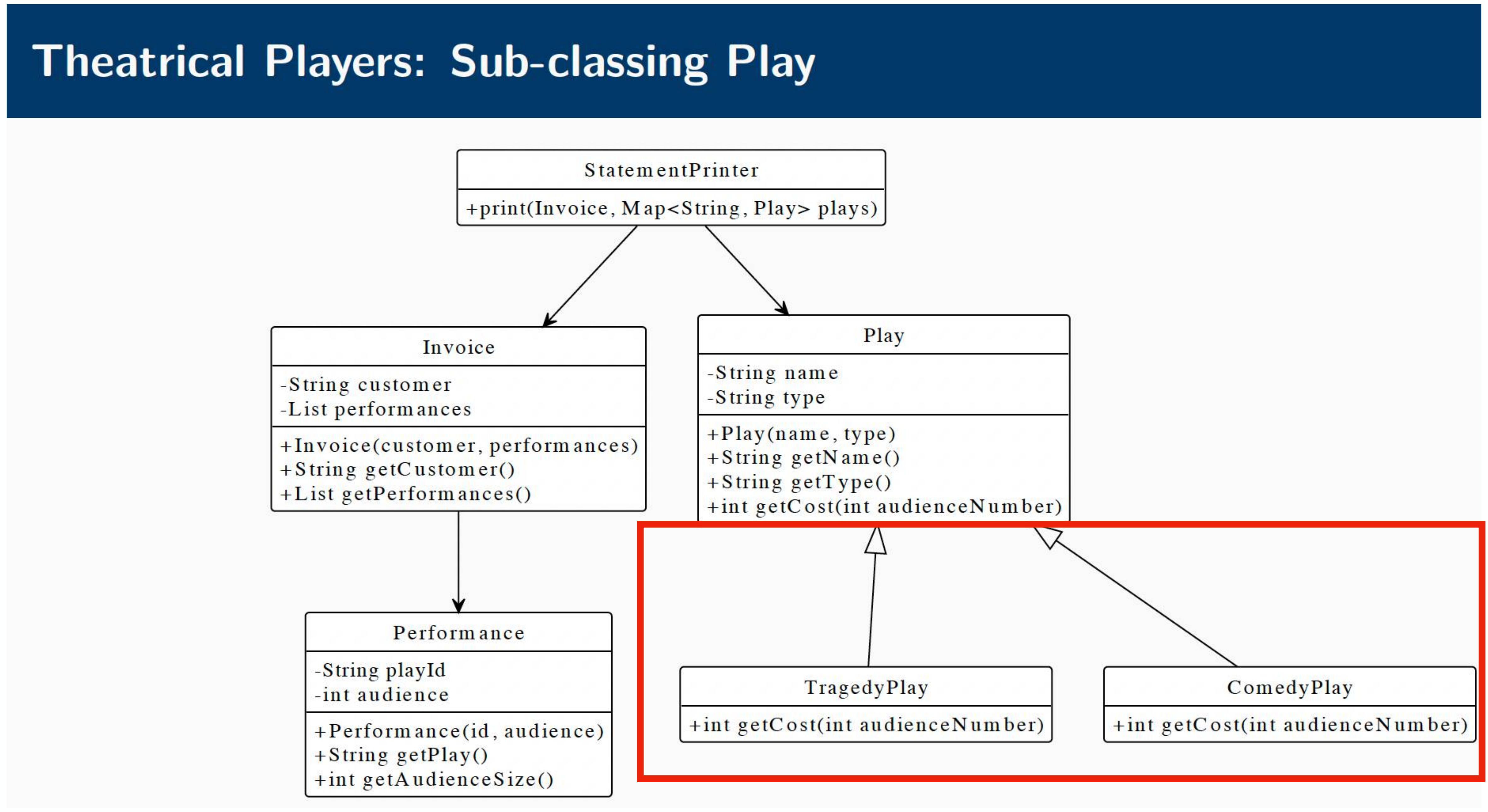
Live Refactoring: Example Improvements

Identify and remove **control coupling**



Theatrical Players: Types as Variables

- `String type` in `Play` is worrying!
  - If the object has a specific type in the real-world
  - Why doesn't it have a type in our program?



Theatrical Players: Sub-classing Play

Student Feedback on Live Refactoring Session

There are
many examples of code
demonstrations in class which
makes it easier for us to
understand the content.

Live coding is really really
helpful no matter for beginner or
advanced programmer

Student Feedback on Live Refactoring Session

There are many examples of code demonstrations in class which makes it easier for us to understand the content.

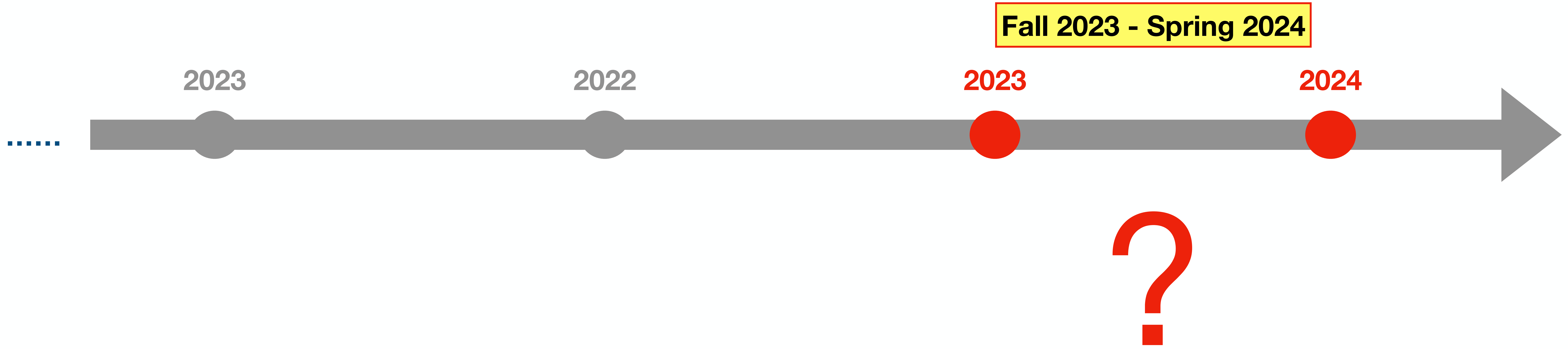
Live coding is really really helpful no matter for beginner or advanced programmer

More Live Coding

[Response to “How could this cause be improved?”]

We did one “live coding” session. We would like to see more live coding session.

Refactoring Code (Low Coupling & High Cohesion)



Refactoring Code (Low Coupling & High Cohesion)

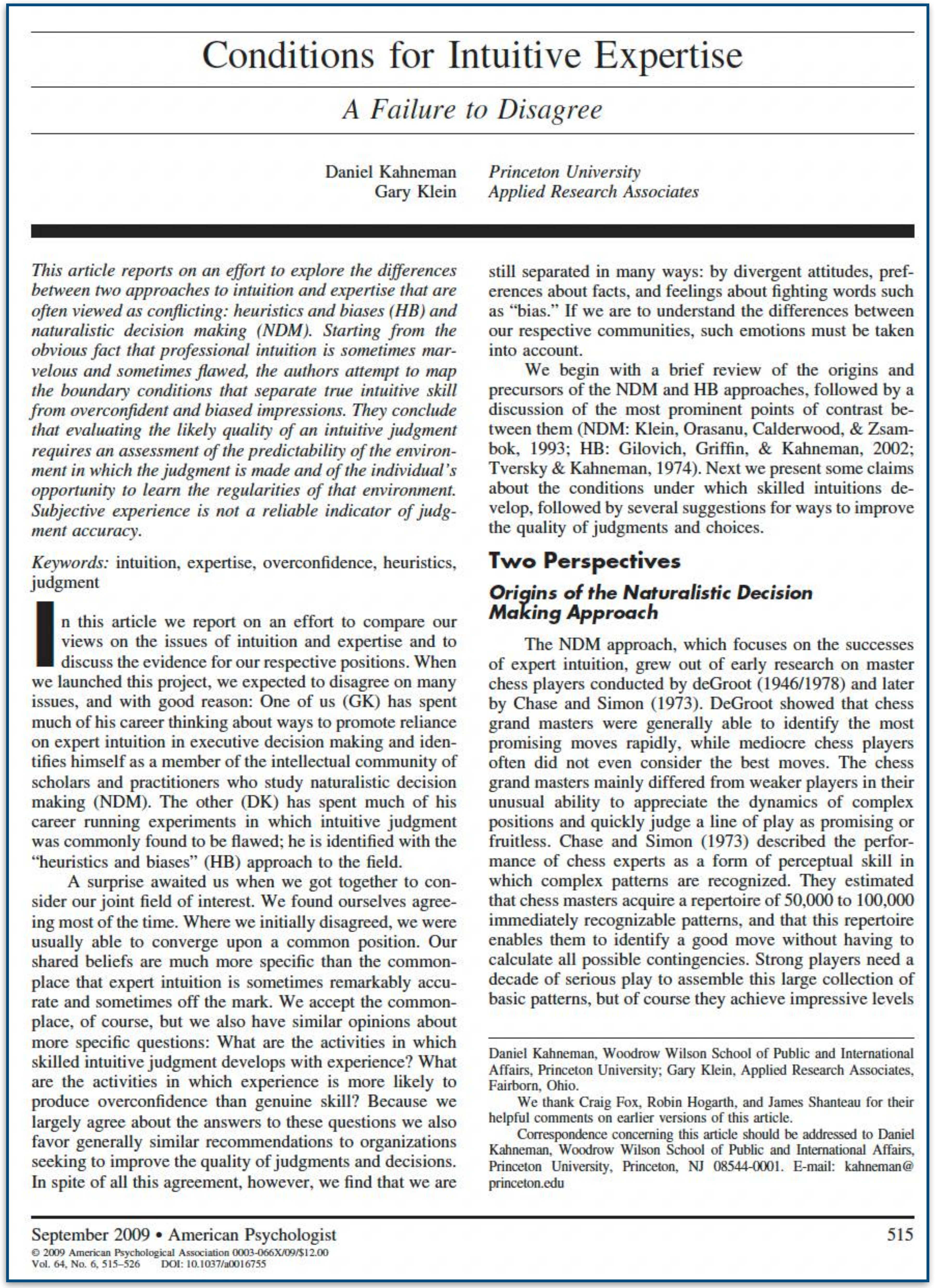
Fall 2023 - Spring 2024
......
2023
2022
2023
2024

How about an **interactive** coding session that employs **scaffolding**?

Interactiveness

Importance of instance feedback in learning:

- Humans are more likely to learn and become experts when the environment is regular (i.e., feedback is *not* delayed or sparse).



Conditions for Intuitive Expertise

A Failure to Disagree

Daniel Kahneman *Princeton University*
Gary Klein *Applied Research Associates*

This article reports on an effort to explore the differences between two approaches to intuition and expertise that are often viewed as conflicting: heuristics and biases (HB) and naturalistic decision making (NDM). Starting from the obvious fact that professional intuition is sometimes marvelous and sometimes flawed, the authors attempt to map the boundary conditions that separate true intuitive skill from overconfident and biased impressions. They conclude that evaluating the likely quality of an intuitive judgment requires an assessment of the predictability of the environment in which the judgment is made and of the individual's opportunity to learn the regularities of that environment. Subjective experience is not a reliable indicator of judgment accuracy.

Keywords: intuition, expertise, overconfidence, heuristics, judgment

In this article we report on an effort to compare our views on the issues of intuition and expertise and to discuss the evidence for our respective positions. When we launched this project, we expected to disagree on many issues, and with good reason: One of us (GK) has spent much of his career thinking about ways to promote reliance on expert intuition in executive decision making and identifies himself as a member of the intellectual community of scholars and practitioners who study naturalistic decision making (NDM). The other (DK) has spent much of his career running experiments in which intuitive judgment was commonly found to be flawed; he is identified with the "heuristics and biases" (HB) approach to the field.

A surprise awaited us when we got together to consider our joint field of interest. We found ourselves agreeing most of the time. Where we initially disagreed, we were usually able to converge upon a common position. Our shared beliefs are much more specific than the commonplace that expert intuition is sometimes remarkably accurate and sometimes off the mark. We accept the commonplace, of course, but we also have similar opinions about more specific questions: What are the activities in which skilled intuitive judgment develops with experience? What are the activities in which experience is more likely to produce overconfidence than genuine skill? Because we largely agree about the answers to these questions we also favor generally similar recommendations to organizations seeking to improve the quality of judgments and decisions. In spite of all this agreement, however, we find that we are still separated in many ways: by divergent attitudes, preferences about facts, and feelings about fighting words such as "bias." If we are to understand the differences between our respective communities, such emotions must be taken into account.

We begin with a brief review of the origins and precursors of the NDM and HB approaches, followed by a discussion of the most prominent points of contrast between them (NDM: Klein, Orasanu, Calderwood, & Zsambok, 1993; HB: Gilovich, Griffin, & Kahneman, 2002; Tversky & Kahneman, 1974). Next we present some claims about the conditions under which skilled intuitions develop, followed by several suggestions for ways to improve the quality of judgments and choices.

Two Perspectives

Origins of the Naturalistic Decision Making Approach

The NDM approach, which focuses on the successes of expert intuition, grew out of early research on master chess players conducted by deGroot (1946/1978) and later by Chase and Simon (1973). DeGroot showed that chess grand masters were generally able to identify the most promising moves rapidly, while mediocre chess players often did not even consider the best moves. The chess grand masters mainly differed from weaker players in their unusual ability to appreciate the dynamics of complex positions and quickly judge a line of play as promising or fruitless. Chase and Simon (1973) described the performance of chess experts as a form of perceptual skill in which complex patterns are recognized. They estimated that chess masters acquire a repertoire of 50,000 to 100,000 immediately recognizable patterns, and that this repertoire enables them to identify a good move without having to calculate all possible contingencies. Strong players need a decade of serious play to assemble this large collection of basic patterns, but of course they achieve impressive levels

Daniel Kahneman, Woodrow Wilson School of Public and International Affairs, Princeton University; Gary Klein, Applied Research Associates, Fairborn, Ohio.

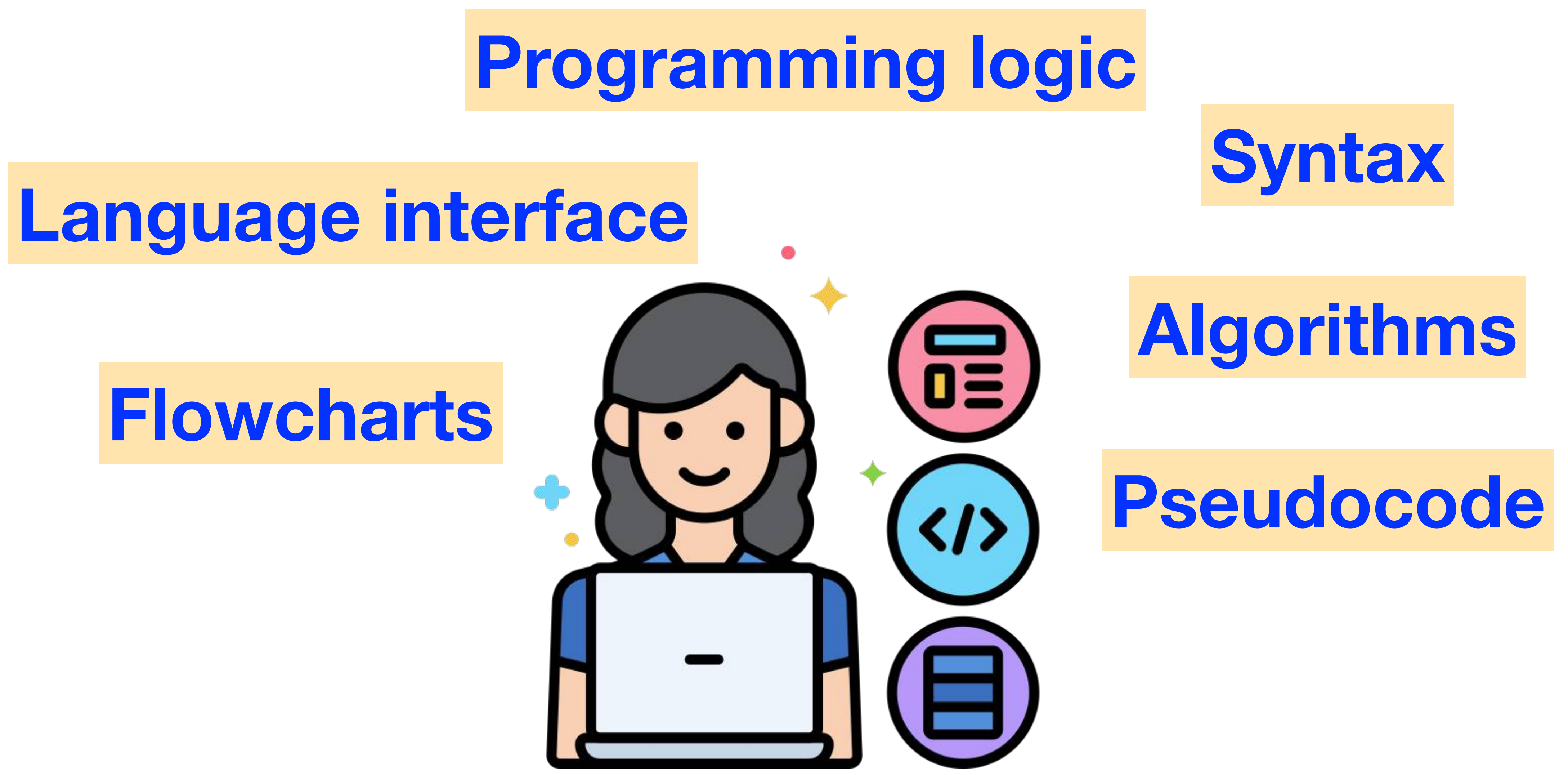
We thank Craig Fox, Robin Hogarth, and James Shanteau for their helpful comments on earlier versions of this article.

Correspondence concerning this article should be addressed to Daniel Kahneman, Woodrow Wilson School of Public and International Affairs, Princeton University, Princeton, NJ 08544-0001. E-mail: kahneman@princeton.edu

© 2009 American Psychological Association 0003-066X/09/$12.00
Vol. 64, No. 6, 515–526 DOI: 10.1037/a0016755

Scaffolding

- One primary obstacle in teaching coding-related activities to novices is teaching syntax[1].
- Scaffolding tools facilitate novices' syntactical proficiency[1].

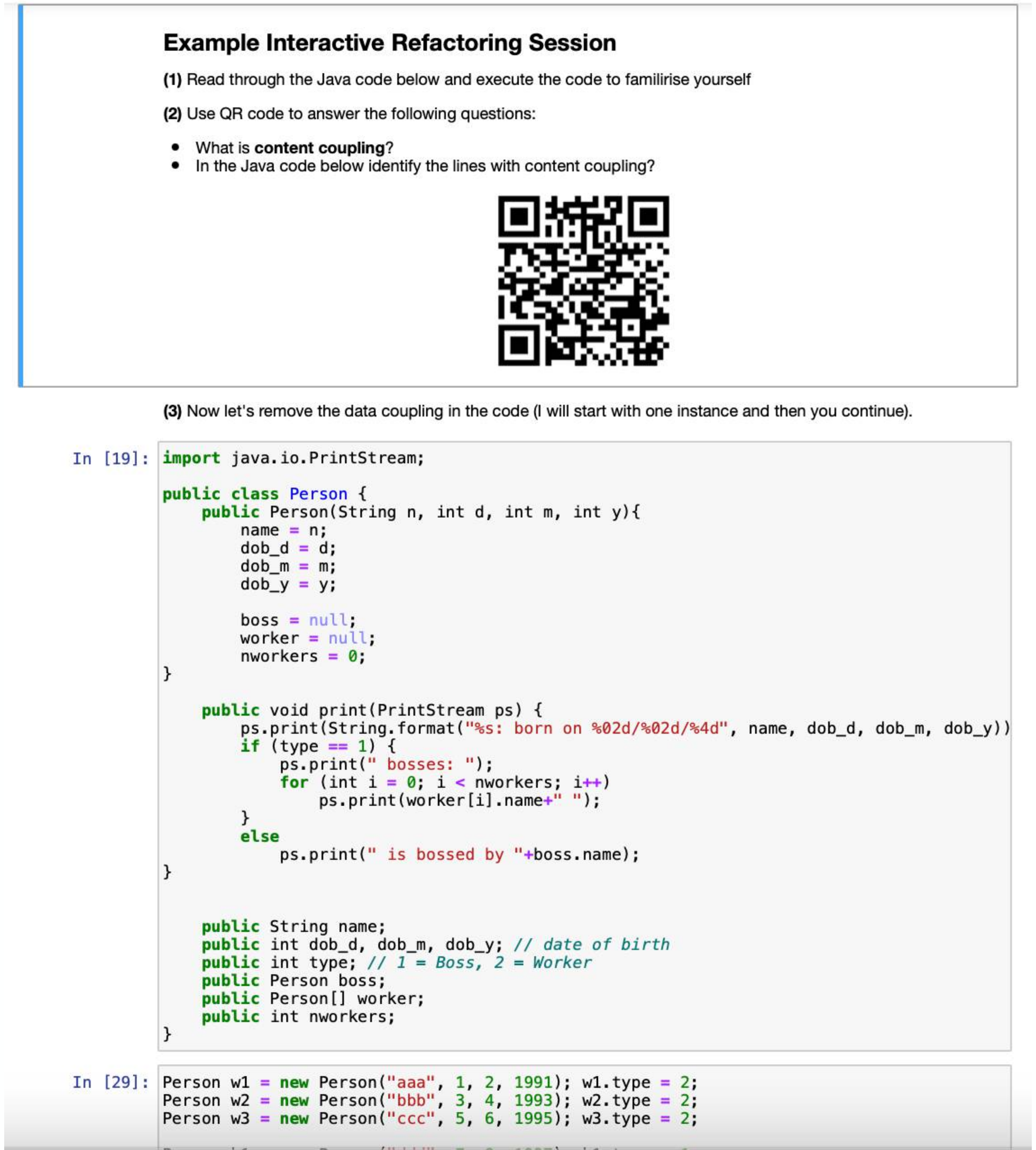


Instructional scaffolding comprises support the teacher gives students throughout the learning process, which the teacher gradually removes as students develop autonomous learning strategies[2].

[1] Al-Imamy, S., Alizadeh, J. & Nour, M. (2006). On the Development of a Programming Teaching Tool: The Effect of Teaching by Templates on the Learning Process. Journal of Information Technology Education, 5,

[2] Beed, P., Hawkins, M., & Roller, C. (1991). Moving learners towards independence: the power of scaffolded instruction. The Reading Teacher, 44(9), 648–655.

Using Jupiter Notebooks to Ensure Interactiveness Employing Scaffolding



Example Interactive Refactoring Session

(1) Read through the Java code below and execute the code to familirise yourself

(2) Use QR code to answer the following questions:

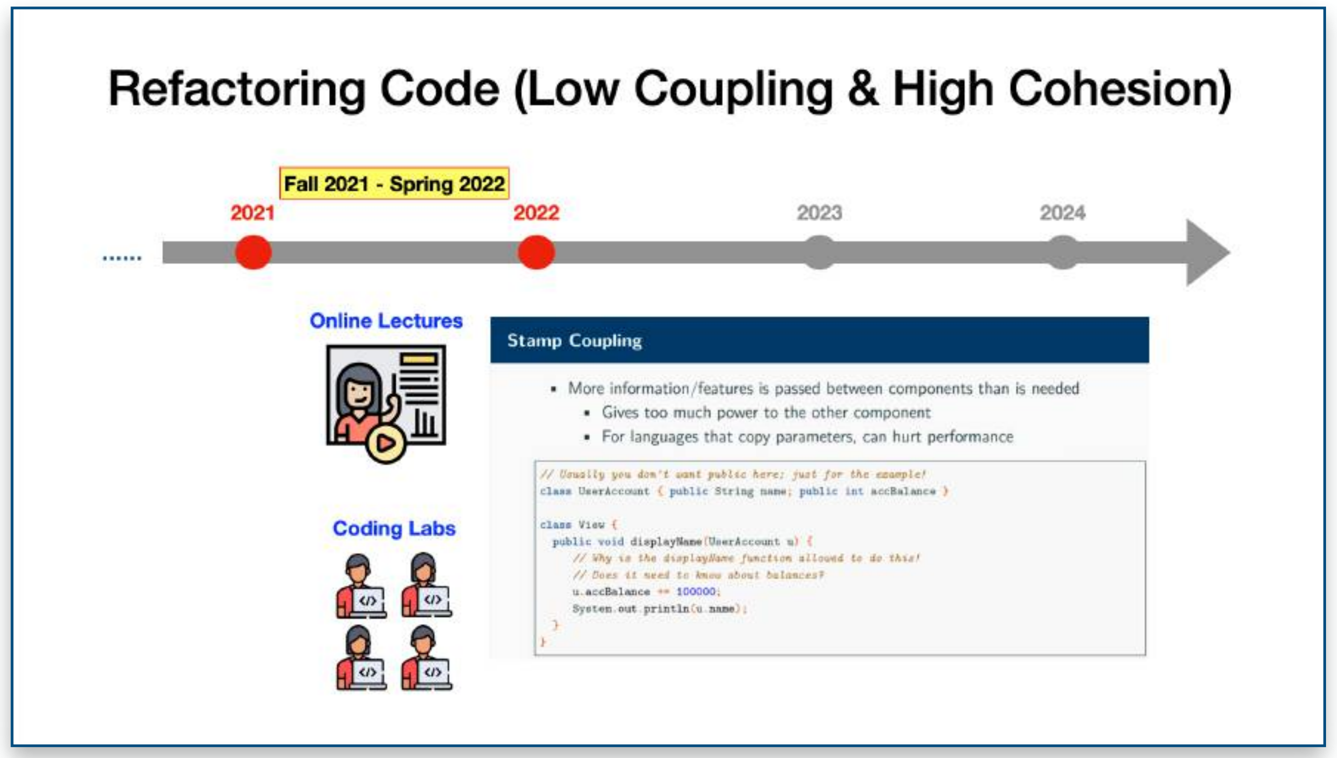
- What is **content coupling**?
- In the Java code below identify the lines with content coupling?

(3) Now let's remove the data coupling in the code (I will start with one instance and then you continue).

In [19]:
```
import java.io.PrintStream;

public class Person {
    public Person(String n, int d, int m, int y){
        name = n;
        dob_d = d;
        dob_m = m;
        dob_y = y;

        boss = null;
        worker = null;
        nworkers = 0;
}

    public void print(PrintStream ps) {
        ps.print(String.format("%s: born on %02d/%02d/%4d", name, dob_d, dob_m, dob_y))
        if (type == 1) {
            ps.print(" bosses: ");
            for (int i = 0; i < nworkers; i++)
                ps.print(worker[i].name+" ");
        }
        else
            ps.print(" is bossed by "+boss.name);
}


    public String name;
    public int dob_d, dob_m, dob_y; // date of birth
    public int type; // 1 = Boss, 2 = Worker
    public Person boss;
    public Person[] worker;
    public int nworkers;
}
```

In [29]:
```
Person w1 = new Person("aaa", 1, 2, 1991); w1.type = 2;
Person w2 = new Person("bbb", 3, 4, 1993); w2.type = 2;
Person w3 = new Person("ccc", 5, 6, 1995); w3.type = 2;
```

increase in task complexity & students' autonomy

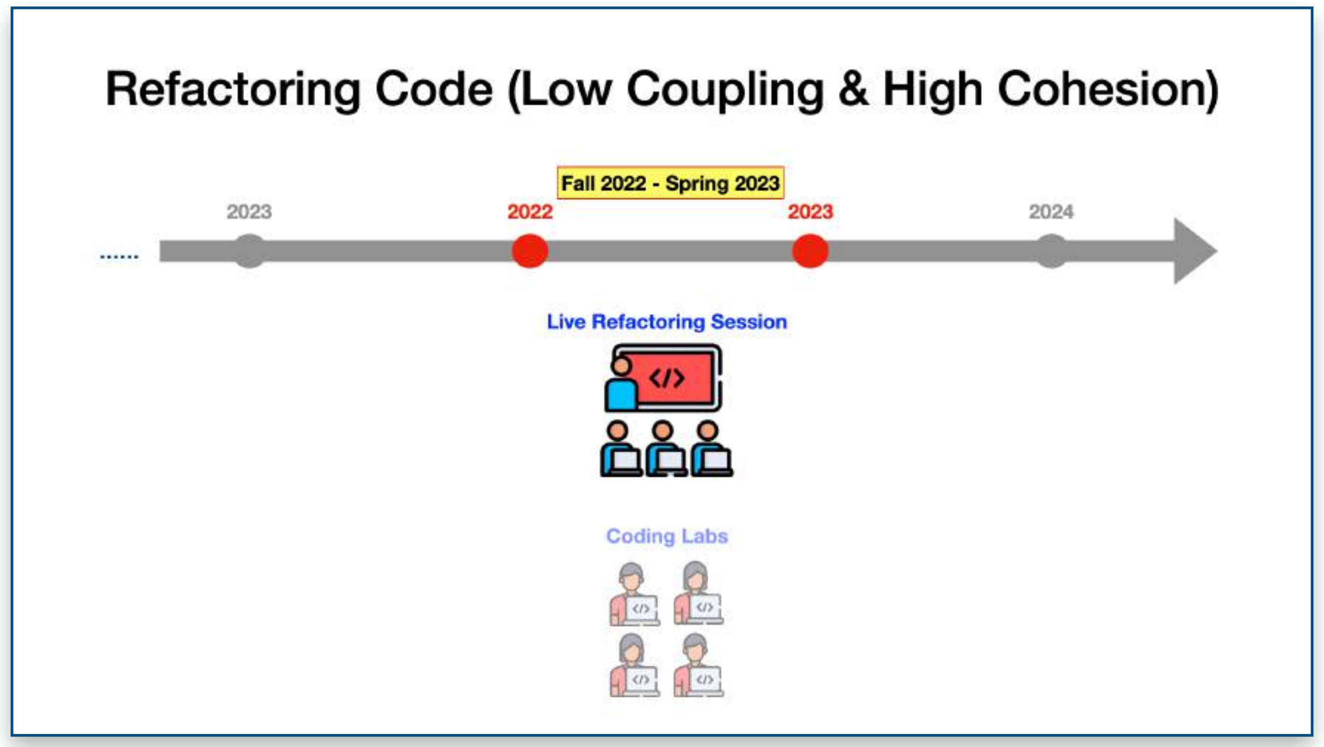


Refactoring Code (Low Coupling & High Cohesion)

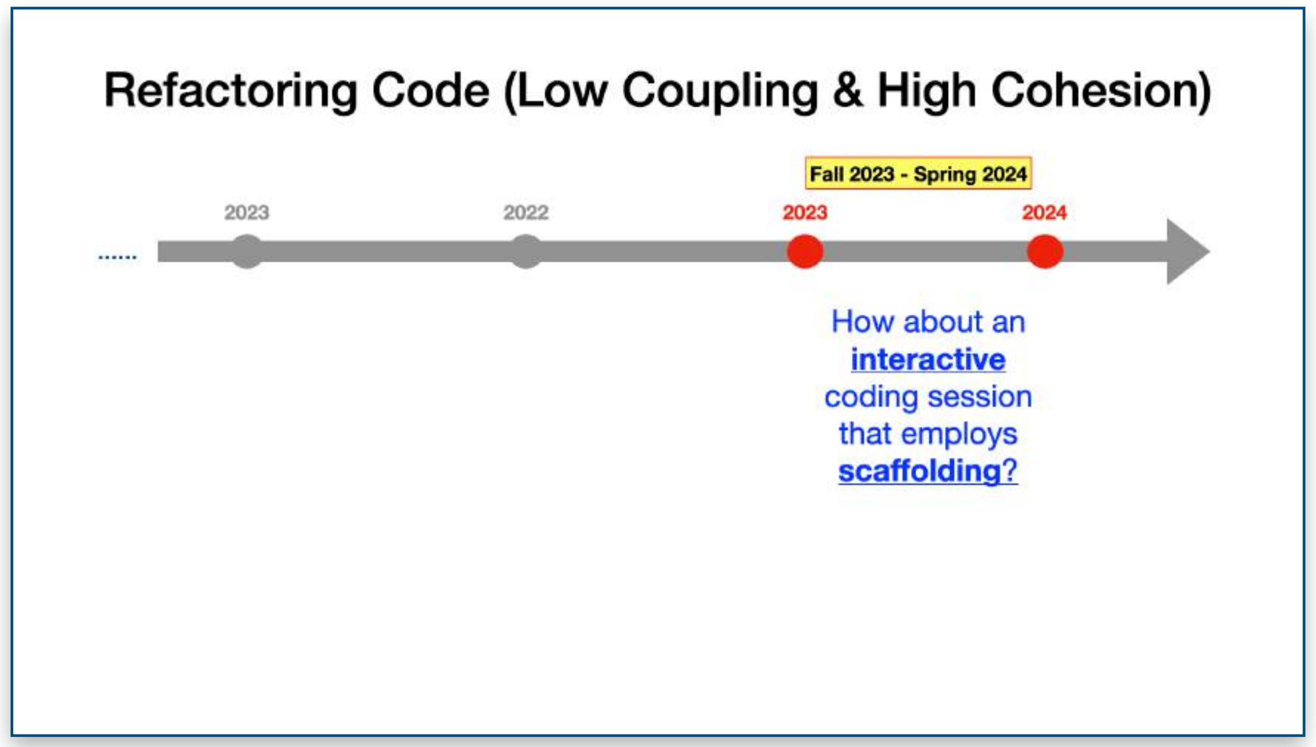
Stamp Coupling

- More information/features is passed between components than is needed
  - Gives too much power to the other component
  - For languages that copy parameters, can hurt performance

```
// Usually you don't want public here; just for the example!
class UserAccount { public String name; public int accBalance }

class View {
  public void displayName(UserAccount u) {
    // Why is the displayName function allowed to do this!
    // Does it need to know about balances?
    u.accBalance += 100000;
    System.out.println(u.name);
  }
}
```

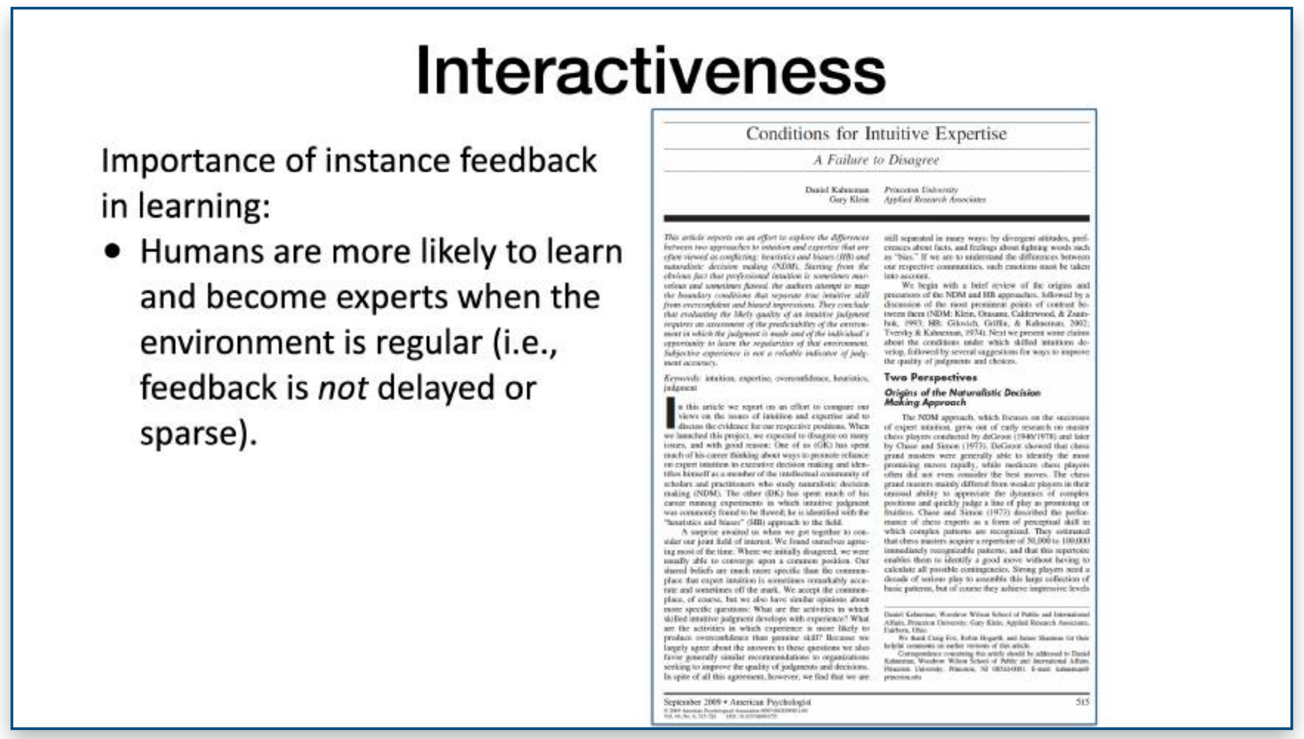


Refactoring Code (Low Coupling & High Cohesion)



Refactoring Code (Low Coupling & High Cohesion)

How about an **interactive** coding session that employs **scaffolding?**

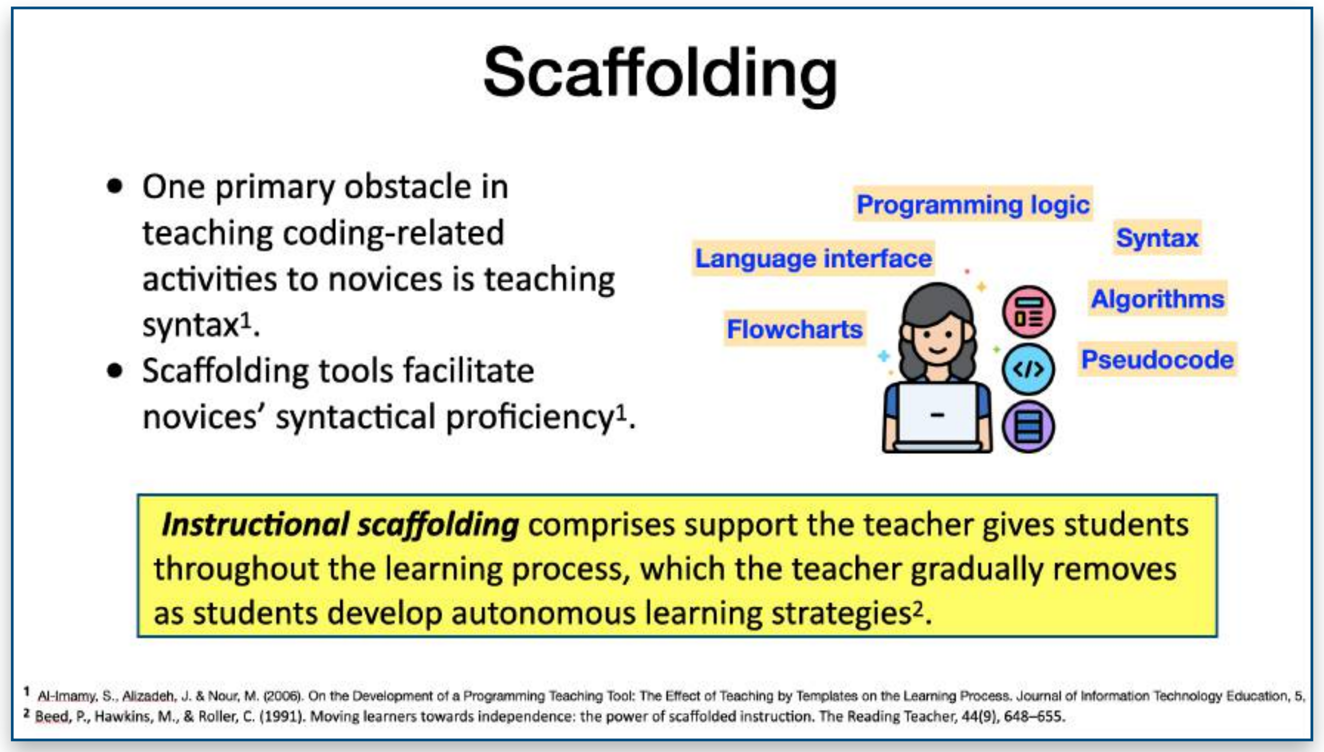


Interactiveness

Importance of instance feedback in learning:

- Humans are more likely to learn and become experts when the environment is regular (i.e., feedback is *not* delayed or sparse).

Conditions for Intuitive Expertise
A Failure to Disagree
Daniel Kahneman, Gary Klein

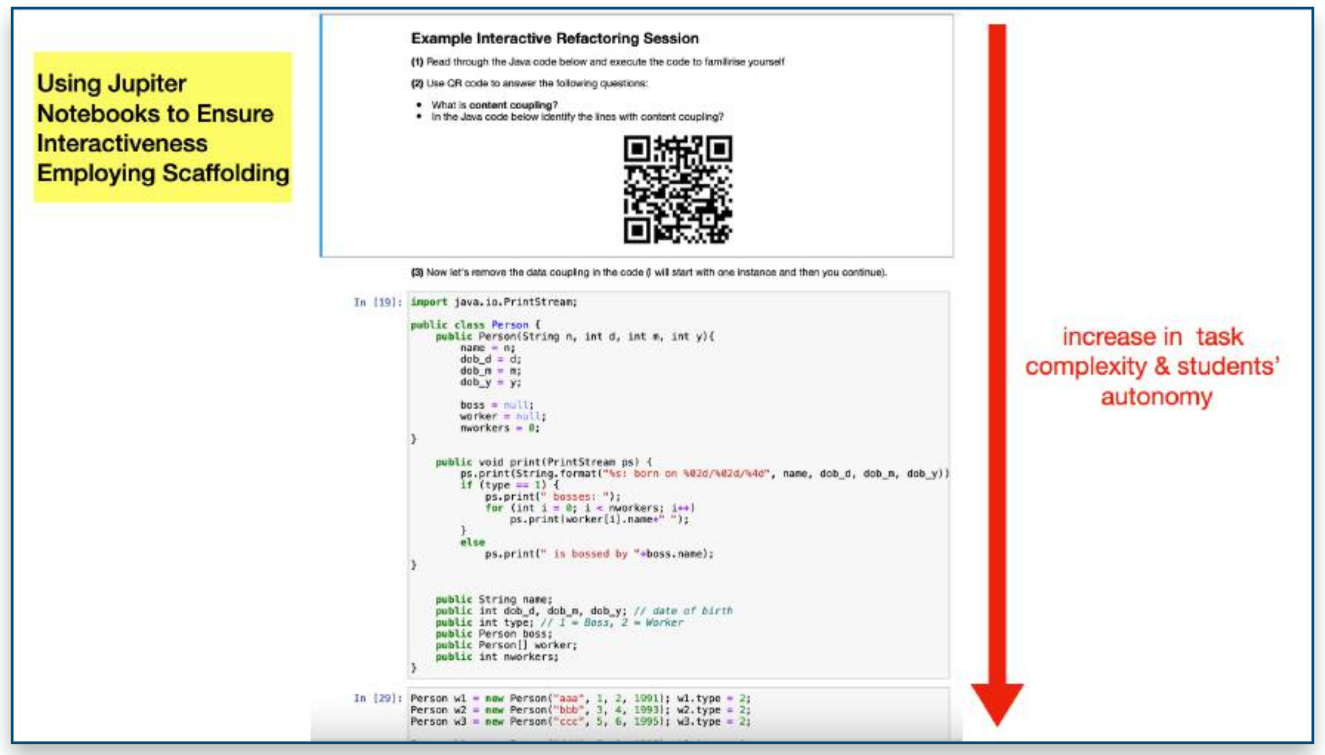


Scaffolding

- One primary obstacle in teaching coding-related activities to novices is teaching syntax[1].
- Scaffolding tools facilitate novices' syntactical proficiency[1].

Instructional scaffolding comprises support the teacher gives students throughout the learning process, which the teacher gradually removes as students develop autonomous learning strategies[2].

[1] Al-Imamy, S., Alizadeh, J. & Nour, M. (2006). On the Development of a Programming Teaching Tool: The Effect of Teaching by Templates on the Learning Process. Journal of Information Technology Education, 5,
[2] Beed, P., Hawkins, M., & Roller, C. (1991). Moving learners towards independence: the power of scaffolded instruction. The Reading Teacher, 44(9), 648–655.



Using Jupiter Notebooks to Ensure Interactiveness Employing Scaffolding

Example Interactive Refactoring Session

(1) Read through the Java code below and execute the code to familirise yourself

(2) Use QR code to answer the following questions:

- What is **content coupling?**
- In the Java code below identify the lines with content coupling?

(3) Now let's remove the data coupling in the code (I will start with one instance and then you continue).

```
In [19]: import java.io.PrintStream;

         public class Person {
             public Person(String n, int d, int m, int y){
                 name = n;
                 dob_d = d;
                 dob_m = m;
                 dob_y = y;

                 boss = null;
                 worker = null;
                 nworkers = 0;
         }

             public void print(PrintStream ps) {
                 ps.print(String.format("%s: born on %02d/%02d/%4d", name, dob_d, dob_m, dob_y))
                 if (type == 1) {
                     ps.print(" bosses: ");
                     for (int i = 0; i < nworkers; i++)
                         ps.print(worker[i].name+" ");
                 }
                 else
                     ps.print(" is bossed by "+boss.name);
         }


             public String name;
             public int dob_d, dob_m, dob_y; // date of birth
             public int type; // 1 = Boss, 2 = Worker
             public Person boss;
             public Person[] worker;
             public int nworkers;
         }

In [29]: Person w1 = new Person("aaa", 1, 2, 1991); w1.type = 2;
         Person w2 = new Person("bbb", 3, 4, 1993); w2.type = 2;
         Person w3 = new Person("ccc", 5, 6, 1995); w3.type = 2;
```

Teaching non-CS Students Software Engineering Basics:

Gül Çalikli, Ph.D.

University of Glasgow

The icons used in this presentation are part of Flaticon

https://www.flaticon.com/